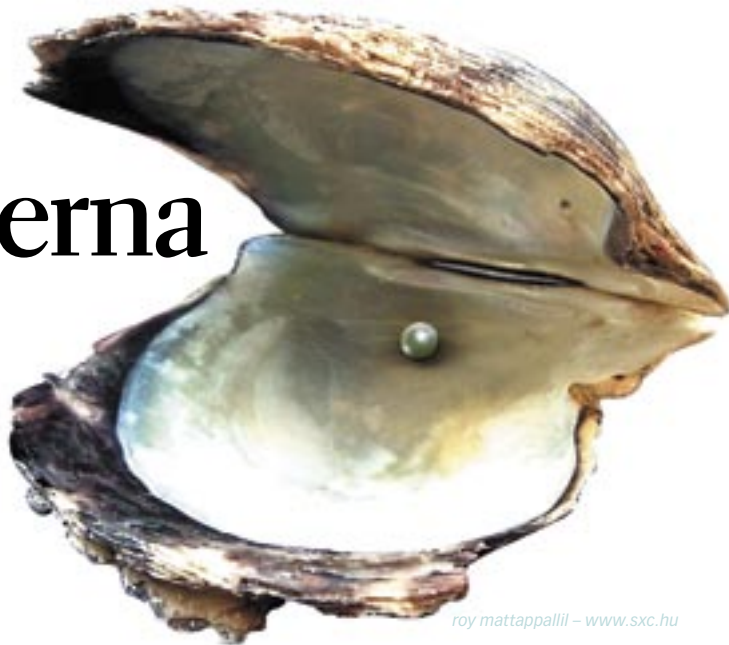


Funções internas do Bash

Eficiência interna

O Bash é o shell padrão de (quase) todas as distribuições Linux. Ele pode automatizar tarefas chamando programas externos em forma de scripts. Mas existem funções internas do próprio Bash que podem tornar seus scripts mais rápidos na escrita e na execução, sem chamar qualquer programa externo.

por Mirko Dölle



roy mattappalil - www.sxc.hu

Antigamente, os *shells* eram incapazes de fazer muito mais do que chamar programas externos e executar comandos internos básicos. Com todos os avanços da última versão do Bash, no entanto, você dificilmente precisará de ferramentas externas.

A principal vantagem das funções internas é que o shell não precisa gerar um

novo processo, o que economiza memória e tempo de processamento. Essa facilidade pode ser importante, principalmente se você precisar rodar um programa como o *grep* ou *cut* num laço, já que o tempo de execução e o consumo de memória de um script podem explodir se você não tomar o devido cuidado. Este artigo descreve algumas técnicas simples para acelerar seus scripts de Bash.

Os seguintes scripts avaliam um arquivo de log do Apache num site simplesmente para comparação. Se você estiver interessado em saber quais páginas foram visitadas, você deve isolar a palavra *GET* no log. Eis um exemplo:

```
84.57.16.30 - - [21/Oct/2005:04:18:26
+0200] "GET /favicon.ico HTTP/1.1" 404
209 "-" "Mozilla/5.0 (X11; U; Linux
i686; de-DE; rv:1.7.5) Gecko/20041122
Firefox/1.0"
```

O **Exemplo 1** mostra uma abordagem usada por diversos scripts de Bash. A chamada do *cat* na **linha 3** primeiro lê o arquivo inteiro, e o despeja no laço *for* como uma lista de parâmetros, o que significa que o Bash tem que guardar o conteúdo do arquivo primeiro. Na **linha 5**, o **Exemplo 1** continua chamando o programa externo *cut* para cada entrada do log. Num Pentium III 750 MHz, esse script demorou mais de 18,5 segundos para examinar um arquivo de log do Apache com 600 KBytes.

Já o **Exemplo 2** demorou apenas 3,3 segundos, quase seis vezes mais rápido:

ela usa o descritor de arquivo 3 para abrir o arquivo, e usa a variável *Request* para processar uma linha por vez num laço, fazendo o Bash examinar e guardar uma única entrada do log. O script então apaga os caracteres do início da linha até *GET* (inclusive) e todos os caracteres desde o fim da linha até *HTTP/* (inclusive).

Seria possível economizar mais um décimo de segundo ao apagar todos os caracteres até aquele em branco desde o fim do pedido, pois assim simplificaríamos a comparação do texto realizada pela função interna do Bash.

Funções de substituição

As ferramentas *basename* e *dirname* podem ser facilmente substituídas usando funções de Bash. Para tal, você precisa das funções de texto `${Variável%Padrão}`, que apaga o menor padrão que casar, a partir do fim do texto, e `${Variável##Padrão}`, que apaga o maior padrão que casar, a partir do início do texto. Somente um *alias* é necessário para substituir o *dirname*:

```
alias nome_do_dir=echo ${1%/*};
```

A funcionalidade para o *basename* não é muito mais complexa, porém você deve considerar o fato de que o *basename* consegue remover a extensão de artigos, o que significa combinar duas funções de texto:

Exemplo 1: Análise de um log com comandos externos

```
01 #!/bin/bash
02
03 IFS=$'\n'
04
05 for l in `cat access.log`; do
06
07     IFS=" "
08
09     echo "${l}" | cut -d" " -f7
10
11 done
```

Exemplo 2: Análise de um log com comandos internos

```
01 #!/bin/bash
02
03 exec 3<access.log
04
05 while read -u 3 Request; do
06
07     Request="${Request##*GET }"
08
09     echo "${Request%% HTTP/*}"
10
11 done
```

```
01 function basename() {
02
03     B=${1##*/}
04
05     echo ${B%$2}
06
07 }
```

É mais eficiente guardar o resultado do texto alterado na variável *B* do que usar o *set* para o primeiro parâmetro e passar o segundo parâmetro.

Mas também não faz sentido substituir cada chamada a programas externos. O **Exemplo 3** é um bom exemplo disso: ela implementa um *grep* rudimentar. Embora o script use apenas algumas linhas de código, e a princípio possa parecer eficiente, ele leva mais de dois minutos para buscar um nome de arquivo num log de servidor web com 600 KBytes – o *grep* realiza o mesmo em menos de um décimo de segundo.

O padrão de busca é o maior assassino de desempenho. Se um pedaço do texto casa com ele, a linha capturada do arquivo é toda deletada por um mecanismo de busca e substituição na **linha 4**. O padrão de busca, `*${1}*`, manda o Bash procurar o padrão em cada caractere de uma linha. Se você usar `##${1}*` como padrão de busca, mandando o Bash começar a busca somente no início da linha, o Bash só fará uma comparação por linha, o que reduz o tempo de execução do script para menos de três segundos. Mas isso ainda é cem vezes mais lento do que o *grep*.

Dissecando IPs

Embora as funções de texto não fossem o gargalo no último exemplo, pode haver outras maneiras mais elegantes e rápidas de se dissecar texto em alguns casos. Por exemplo, se você precisa ordenar os endereços IP de onde pedidos se originaram, tanto o *sort* quanto uma função simples de ordenação léxica serão incapazes de ajudar. Esse procedimento retornaria `217.83.13.152` antes de `62.104.118.59`. Em vez disso, será necessário extrair os bytes individuais do número IP, convertê-los para um formato ordenável, e então mostrar os resultados sem repetição.

Os **Exemplos 4 e 5** mostram duas formas possíveis com desempenhos completamente diferentes. O script do **Exemplo 4** começa examinando o arquivo de log linha por linha (**linha 3**), e depois extrai o primeiro IP na **linha 4**. As **linhas 6 a 10** apagam um octeto por vez a partir do fim do número IP, e guarda o byte do endereço como um decimal no vetor *IP*.

A chamada na **linha 11** ao *printf*, que também é um comando interno do Bash, serve para jogar na saída os quatro octetos do endereço IP como decimais de três dígitos separados por zero e sem preenchimento. A última linha faz um *pipe* da saída do comando externo *sort*, antes que o *uniq* retire os números duplicados. O **Exemplo 4** leva aproximadamente 2,6 segundos para processar um arquivo de log do Apache com 600 KBytes.

Funções para compactação

O programa do **Exemplo 5** faz o mesmo que o do **Exemplo 4**, mas leva apenas 1,6 segundos, uma melhora de quase 40%. As funções para texto das **linhas 4 a 10** do **Exemplo 4** são os responsáveis pela diferença: em vez de extrair o IP primeiro e só então usar sete funções diferentes para dissecá-lo, o **Exemplo 5** chama a função *read* interna do Bash com a variável *IFS*. O Bash trata os caracteres armazenados em *IFS* como separadores de parâmetros – por padrão, são os caracteres de: espaço, tabulação e nova linha.

A **linha 3** do **Exemplo 5** define os caracteres de ponto e espaço em branco como separadores. Chamar *read* com a opção *-a* manda a função não armazenar a linha inteira na variável, mas usar o separador a partir de *IFS*, e gravar a entrada na variável de vetor *IP*, um elemento de cada vez. Isso manda os octetos que compõem o IP direto para as variáveis *IP[0]* a *IP[3]* ao chamar *read*. O conteúdo dos elementos do outro vetor não é importante agora. Portanto, uma simples chamada de função no **Exemplo 5** substitui as **linhas 3 a 10** do **Exemplo 4**.

Infelizmente, apesar de ser possível substituir as chamadas aos comandos externos *sort* e *uniq* por funções de Bash correspondentes, não se pode esperar que um script de Bash alcance a eficiência do *sort*, um programa escrito em C. Assim como na vida real, alguns de nossos esforços de sintonia fina em Bash podem não melhorar a velocidade do script, mas programar com vistas à eficiência continua sendo um bom negócio. ■

Mais Informações

[1] Programas de exemplo: <http://www.linux-magazine.com/Magazine/Downloads/64/bash/>

Exemplo 3: Grep interno

```
01 #!/bin/bash
02
03 exec 3<&$2
04
05 while read -u 3 line; do
06
07     if [ -z "${line/*${1}*}" ]; then
08
09         echo "$line"
10
11     fi
12
13 done
```

Exemplo 4: Funções para texto

```
01 #!/bin/bash
02
03 function GetIP() {
04
05     while read -u $1 Request; do
06
07         tmp="${Request%% *}"
08
09         IP[1]="${tmp%.*}"
10
11         IP[4]="${tmp##*}"
12
13         tmp="${tmp%.*}"
14
15         IP[3]="${tmp##*}"
16
17         tmp="${tmp%.*}"
18
19         IP[2]="${tmp##*}"
20
21         printf "%03d.%03d.%03d.%03d\n" ${IP[1]} ${IP[2]}
22     ➔ ${IP[3]} ${IP[4]}
23     done
24
25 }
26
27 exec 3<access.log
28
29 GetIP 3 | sort | uniq
```

Exemplo 5: Funções para texto

```
01 #!/bin/bash
02
03 function GetIP() {
04
05     IFS="."
06
07     while read -u $1 -a IP; do
08
09         printf "%03d.%03d.%03d.%03d\n" ${IP[0]} ${IP[1]}
10     ➔ ${IP[2]} ${IP[3]}
11     done
12
13 }
14
15 exec 3<access.log
16
17 GetIP 3 | sort | uniq
```